# Homework 3

1. Gadget Testing
   1. Brute Force

This algorithm would be to simply starting at level 1 and going up 1 level at a time, drop the gadget and stopping when it breaks.

Pseudo Code:

Floor = 0

While (notBroken) :

Floor += 1

dropGadget(floor)

Return floor

Worst Case: Gadget is unbreakable and you need to drop from every floor of building during testing (time = n)

* 1. Optimization (better than n (ie log n or constant)

The help of a second gadget at first thought would be to begin cutting n in half. Then we could start at n/2nd floor, drop gadget, if breaks then brute force bottom half. If not, then try the middle of the top half and so on.

Pseudo Code:

NumFloors = n // number of floors to be checked

G1Floor = 0

G2Floor = numFloors /2 // half way point

While (g2 != broken):

dropGadget(g2, g2Floor)

if (g2 != broken):

g1Floor = g2Floor

temp = (numFloor – g2Floor) / 2

g2Floor += temp // only need to go up if unbroken

if g2Floor = numFloor

unbreakable gadget

Reapeat first Algo from g1 -g2 floor once broken

Time Complexity:

Average would be (log n) since we are potentially cutting number of trials in half for each attempt with g2 gadget, with worst case still being n (though n/2, still linear time)

T(n) = T(n/2) + 4 x=n/2 t(n) = t(n/4) +8 x= n/4 t(n) = t(n/8) + 12

T(n) = T(n/2^k) + 4k T(1) = c : n = 2^k k = log(n)

T(n) = T(n/n) + 4log(n) = c + 4 log(n)

1. Counting Context
   1. Brute Force

My brute force algorithm would find an A in the string then add the number of B’s after to a counter, then find the next A and do it again and again.

Pseudo Code:

Counter = 0

For letter in InputString:

If letter == A:

From current to end of list

If current == B:

Counter += 1

Worst Case

T(n) = = = (n+1) (n(n+1)/2) in Θ(n^3)

Best Case (no A found)

Θ(n)

* 1. Optimization

Thoughts on more efficient: indexes (no, still nested for loop), Find B’s and look Back (Would be same as A’s but Best Case would be no B’s instead), Many languages could use predefined func to get all combos of A (\*) B then just return the length but not our algorithm.

Psuedo Code:

Not sure what would be better.

1. Graph (figure 1)
   1. Adjacency Matrix and list (assuming no self loops)

Adjacency Matrix

A B C D E F G

A 0 1 1 1 1 0 0

B 1 0 0 1 0 1 0

C 1 0 0 0 0 0 1

D 1 1 0 0 0 1 0

E 1 0 0 0 0 0 1

F 0 1 0 1 0 0 0

G 0 0 1 0 1 0 0

Adjacency List

A -> B, C, D, E B -> A, D, F C -> A, G D-> A, B, F

E -> A, G F -> B, D G -> C, E

* 1. Start at A, DFS

Push A, Push B, Push D, Push F, Pop F, Pop D, Pop B, Push C, Push G, Push E, Pop E, Pop G, Pop A

* 1. Start at A, BFS

Queue:

A, B, C, D, E, F, G

A adds b,c,d,e then B adds f, then C adds g and all nodes are seen

Tree: (Assume arrows going down the tree)

A

B C D E (all added from parent node A)

F (added by B)

G (added by C)

1. Knapsack
   1. Write pseudo code and determine efficiency
   2. Consider given scenario
   3. Prove greedy works
   4. See Code from HW3 Code assignment